VIPER

Novemmber - December 1983
Volume 5, Number 4

Journal of the VIP Hobby Computer Assn.

The VIPER was founded by ARESCO, Inc. in June 1978

****

PIPS for VIPS IV - Part 4

This issue of VIPER continues the serialization of Tom Swan's PIPS for VIPS IV. The first section is called "A Binary Tree for the Holidays" and in the original manuscript followed the Cos-Melodeon program. It was moved to this issue to try to match the calendar.

Page 5.04.06 continues the text which was printed in the preceeding VIPER, so it should follow 5.03.25 to complete that section. The appendix pages (and those from the issue before 5.03) are intended to go at the very end of your loose-leaf notebook when you have received everything.

Also in this issue on page 5.04.12 is an interesting method of generating a two-color display with your VIP by Bob Casey, WA2ISE. Bob's circuit uses 4 chips and 1 transistor to create yellow dots on a blue background.

Cassettes of PIPS IV are still available and the cost for the ten programs on the tape is $5. You may send your check to VIPHCA at the usual address. Each program on the tape is complete, with the appropriate CHIP-8 interpreter, where necessary. The program titles are:

1. Cos Melodeon
2. Holiday tree
3. Attack of Micromen
4. Line up
5. Sweeper
6. Move on
7. Box
8. Shape maker
9. Graphics #1
10. Graphics #2

NEWS & VIEWS

First of all, I have to appologize to Jerry Krizek who had sent in an advertisment for this issue of VIPER. Somehow or other when I went to gather the material for this issue, his ad was not to be found. Still have the envelope, but the contents cannot be found. He has a bunch of VIP compatible memory boards available at close-out prices. If you are really anxious for a memory board, drop him an SASE and I'm sure he'll send you the particulars right away. Hopefully, I'll get my act together and you will be able to see the ad in next issue. His address is: 722 N. Morada Ave. West Covina, CA 91790.

The small computer market is still in a state of flux, as I'm sure you are aware. Texas Instruments has thrown in the towel on its TI-99 line. You can still get them at many locations for around $50, and at that price, it's an interesting machine. But the only software of consequence for the '99 is made by TI and in many cases, the ROM cartridges cost more than the computer!

Timex has had some problems getting their new color computer, the 2068 to market. Some areas have them, many do not. The model 1500, however is available in many areas at around $75 or so. That machine is an upgraded version that is software and hardware compatible with the earlier ZX-81 and TS-1000 computers. Radio Shack has introduced a new line of TRS-80 Color computers, including the "Micro" Color Computer, the MC-10. And Commodore has been selling C-64's as fast as they can be made.

All of these small computers are supported well by the manufacturers and outside companies who make programs and accessories for the machines. Indeed, the manufacturers have seemed to bend over backwards to assist other companies who make software and hardware for their machines. Maybe that's why they are all selling so well. And each of the above machines has SEVERAL magazines supporting just that one particular computer or family of computers. The magazines are especially useful sources of information about those machines. Sadly, though, none of these popular "little" machines uses an 1802 processor. And that's a shame. The 1802 could certainly do as good a job as the 6502 in the Commodores and Ataris.

Let me take this moment to wish all of you an enjoyable holiday season and best wishes for a happy 1984. George Orwell, here we come!!

73,

Ray

A BINARY TREE FOR THE HOLIDAYS

Having no idea what time of year it is while you are reading these words, I realize I may be way out of season with the following program. So if it happens to be the dead of summer, my apologies. In that case, load 7 pages from tape, think December and flip to run.

* * *

Christmas trees are a premium item here in Mexico and I was thinking of our first Mexican Christmas tree when I wrote this program. Trees may be found in the market but only for outrageous prices. In fact any wood product, especially from hardwoods, is very expensive. There just aren't that many good trees in this hot country.

A week before Christmas 1978 my friend Jose Luis showed up at the door with a misshapen, scraggley little cedar tree which he had carried from the mountains five miles on his back so we would have a Christmas tree! As my mother says every year, "It was the most beautiful tree we ever had." But that's not the end of the story.

Our apartment is several stories above a little corn field which is above the road to Acapulco. In fact everything in these mountains seems to be above something else. It's like

living in an airplane.

A week after Christmas our little cedar was turning brown, and with the city trash collection idiosyncracies around here I wasn't quite sure how to dispose of it. One day Jose came in, and I said would he get rid of the tree for us? Sure, he said, no problem. So I removed the base, gave him the dried remains of his present and he promptly walked over to the open window and tossed it out! All done, said my friend dusting his hands and waving adios on his way out the door! I thought my gaping mouth would never close.

When appropriate, "Have a Merry!"

```
        ; **************
        ; * CHRISTMAS *
        ; *  1 9 7 9  *
        ; **************

        ; USES:
        ;  V0,V1,V2,VA,
        ;  VC,VD,VE,VF,
        ;  VB

        ; REQUIRES:
        ;  4-PAGE HI-RES
        ;  CHIP-8 LANG.

        ; C.1980 T. SWAN

0500 BEGIN :2518  TREE
  02       2550  TRUNK
  04       2560  STAR
  06       256A  LITES

0508       6B02 ;VB=2
  0A TRS1  :25E8  BLNKR
  0C       261A  CHOO

050E       7BFF ;VB-1
  10       3B00 ;SK=0
  12       150A  TRS1
  14       0200 ;CLEAR
  16       1500  BEGIN

        ;END MAIN LOOP

0518 TREE  :6010 ;V0=10
  1A       616E ;V1=6E
  1C       622F ;V2=2F
  1E TREE1 :252C  BRNCH
0520       7001 ;V0+1
  22       72FF ;V2-1
  24       310E ;SK=0E
  26       151E  TREE1
  28       00EE ;RET

        ;END TREE SUB

052A LEAF  :8000 ;1 BIT
  2C BRNCH :6E06 ;VE=2
  2E       A52A  LEAF
0530 BRNC1 :8C00 ;VC=V0
  32       8D10 ;VD=V1
  34 BRNC2 :DCD1 ;SHOW
  36       7C01 ;VC+01
  38       5C20 ;2=C?
  3A       1534  BRNC2
  3C       71FF ;V1-1
  3E       7EFF ;VE-1
0540       3E00 ;VE=0?
  42       1530  BRNC1
  44       00EE ;RET

        ;END BRNCH SUB
```

```
0546 ROOTS :3030
  48       3030
  4A       FCFC
  4C       FCFC
  4E       FCFC
0550 TRUNK :A546  ROOTS
  52       6C1D ;VC=1D
  54       6D70 ;VD=70
  56       DCDA ;SHOW
  58       00EE ;RET

     ;END TRUNK SUB

055A STAR1 :2020
  5C       7020
  5E       2000

0560 STAR  :A55A  STAR1
  62       6C1D ;VC=1D
  64       6D05 ;VD=5
  66       DCD5 ;SHOW
  68       00EE ;RET

     ;END STAR SUB

056A LITES :6E00 ;VE=00
  6C LITE1 :A590  HOOKS
  6E       FE1E ;I+VE
0570       F265 ;GET
  72       A584  BULBS
  74       F21E ;I+V2
  76       D013 ;SHOW
  78       6A10 ;VA=10
  7A       25DE  TIMER
  7C       7E03 ;VE+3
  7E       3E4E ;SK=4E
0580       156C  LITE1
  82       00EE ;RET

     ;END LITES SUB

0584 BULBS :8000 ;  1
  86       0000

  88       8080 ;  2
  8A       8000

  8C       40E0 ;  3
  8E       4000

     ;END BULBS
```

```
0590 HOOKS :1C2A ;STRNG
  92       041E
  94       3204
  96       2136 ;  1
  98       0424
  9A       3804

  9C       1A3E ;STRNG
  9E       041C


05A0       4608
  A2       214A ;  2
  A4       0424
  A6       4E04
  A8       2750
  AA       0416
  AC       5208 ;STRNG
  AE       1B58
05B0       041F
  B2       5C04 ;  3
  B4       235E
  B6       0829
  B8       6004

  BA       1C03 ;LIGHT
  BC       0022
  BE       0300
05C0       1B0A ;RAYS
  C2       0023
  C4       0A00
  C6       1900
  C8       0025
  CA       0000
  CC       180E
  CE       0027
05D0       0E00
  D2       1512
  D4       002A
  D6       1200
  D8       1216
  DA       002D
  DC       1600

     ;END HOOKS

  DE TIMER :FA15 ;T=VA
05E0 TIME1 :FA07 ;VA=T
  E2       3A00 ;VA=0?
  E4       15E0  TIME1
  E6       00EE ;RET

     ;END TIMER
```

```
   E8 BLNKR  :6D80 ;VD=80
   EA BLNK1  :CE1F ;RNDOM
   EC         6A19 ;LIMIT
   EE         8AE5 ;VA-VE
 05F0         3F01 ;VF=1?
   F2         15EA  BLNK1
   F4         8AE0 ;VA=VE
   F6         8EE4 ;VEx3
   F8         8EA4 ;  "
   FA         A590  HOOKS
   FC         FE1E ;I+VE
   FE         F265 ;GET
 0600         A584  BULBS
   02         F21E ;I+V2
   04         D013 ;SHOW
   06         6A10 ;VA=10
   08         25DE  TIMER

 060A         D013 ;OFF
   0C         6A04 ;VA=4
   0E         25DE  TIMER
 0610         7DFF ;VD-1
   12         3D00 ;VD=0?
   14         15EA  BLNK1
   16         00EE ;RET

    ;END BLINKER

 0618 TRAIN  :8080

   1A CH00   :6D08 ;VD=08
   1C CH001  :6E00 ;VE=0
   1E CH002  :A64A  TRACK
 0620         FE1E ;I+VE
   22         F165 ;GET
   24         A618  TRAIN
   26 CH003  :D012 ;SHOW
   28         6A08 ;VA=8
   2A         25DE  TIMER
   2C         3F00 ;HIT?
   2E         1626  CH003
 0630         CA0F ;RNDOM
   32         4A00 ;SK≠0
   34         2644  TOOT
   36         7E02 ;VE+02
   38         3E1A ;SK=1A
   3A         161E  CH002
   3C         7DFF ;VD-1
   3E         3D00 ;SK=0
 0640         161C  CH001
   42         00EE ;RET

    ;END CH00-CH00
```

```
 0644 TOOT   :6A40 ;VA=40
   46         FA18 ;TONE
   48         00EE ;RET

    ;END TOOT SUB

 064A TRACK  :1972
   4C         1774
   4E         1678
 0650         177C
   52         1A7E
   54         1D7E
   56         207E
   58         237E
   5A         267E
   5C         297C
   5E         2A78
 0660         2974
   62         2772

    ;END TRACK

    ;END PROGRAM
```

your application.

The AND instruction at $012A is there as a bit test giving a program an easy way to test if a bit was already on at that X, Y coordinate. If the bit was equal to one, RE.0 will not be equal to zero on return from POINT. If RE.0 is zero on return, then that bit was not on. This is similar to the VF hit indicator in CHIP-8 except that the hit value may be any value not equal to zero to signal hits. In CHIP-8 VF is always set to one, an assumption you must not make when using POINT.

USE AND CARE OF POINT

POINT knows where to begin the display matrix by the address contained in RB.1. Regardless of the resolution being used, this matrix will always begin at a page boundary with the low byte of the address equal to zero. Room has been left by the NOP's (SEX R2 instructions are often used as NOPS as X is usually equal to 2 anyway) at addresses $0121 - $0122. When using page switching, these two instructions are changed to cause POINT to operate on the off screen display page. (see FLOP sub.)

To display a point, the following steps must be executed.

1) Set RE.0 = X coordinate from 00 to $3F
2) Set RE.1 = Y coordinate from 00 to $7F
3) CALL POINT

No checks are made for out of range points and it is the program's responsibility to be sure that X and Y are within the above values. This is especially important when using POINT with low resolution interrupts as display information could be written into unwanted areas of memory outside of the address boundaries of the display refresh. Keeping such checks out of the sub makes it go faster. You could also eliminate the "hit" test for an additional, but very slight, speed increase.

POINT is crucial to the graphics package. Without this sub, LINE, SHAPE, and PLOT would cease to function -- they all ultimately use POINT to animate the computer display. Besides if it wasn't here, there would be no POINT to this book.

CLEAR THE AIR

There really isn't too much to say about this subroutine located at $0130. Calling CLEAR will simply set the entire display refresh page (four memory pages remember) to zero bytes. You do not have to give any information to the sub except that as in POINT, RB.1 must be set to the address of the display refresh memory area.

When using this sub with lower resolution displays, it will still work equally well. However, CLEAR always erases four 256 byte blocks of memory regardless of how much of that memory is being used for the display. Take care that this does not include part of your program!

You may adjust CLEAR to erase less (or more) memory by changing the bytes at the following locations:

```
$0132  XX  ;Number of pages to erase minus one
$0138  XX  ;Number of pages to erase (00 = 256 pages)
```

CLEAR is optimized for speed -- the entire four page display refresh appears to vanish almost at once, a helpful feature when not using page switching for animations. As with POINT, changes to CLEAR must be made when using the page switching technique. These changes are detailed in the remarks on using the FLOP sub.

Hope that CLEARS things up for you.

TOE THE LINE

Most graphics systems contain a way to connect two points with a straight line. Because the display is like a blocked sheet of graph paper, however, only horizontal and vertical lines come out really straight. Diagonal lines are necessarily only approximations of straight lines and appear "stair stepped" and jagged, but less so in high resolution displays. Deciding which points are closest to the slope of the line is the job of the LINE subroutine.

The first step is to "normalize" the line given two pairs of X,Y coordinates marked as X_0, Y_0 (from point) and X_1, Y_1 (to point). The absolute values of $(X_0 - X_1)$ and $(Y_0 - Y_1)$ are placed

in R9.1 and R9.0 labeled XN and YN in the listing. Finally XN is assured of being greater or equal to YN. The purpose of all this is to take any of the eight possible directions a line could drawn (four "straight" lines and four diagonals) and transform the line as if it were being drawn from $X_0 = 0$, $Y_0 = 0$ to a positive X_1, Y_1 with $X_1 \geq Y_1$.

If you visualize such a line, say from 0,0 to 20,10, you should see that there must be exactly $X_1 + 1$ points on the line when $X_0 = 0$. That is, we simply need to calculate Y_0 for X_0, $X_0 + 1$, $X_0 + 2 \ldots X_0 + N$ while holding the course as closely as possible to the true line. If when increasing X_0 we move too far off the line, then Y_0 must also be increased. As long as X_0 has not gone too far, then Y_0 remains the same resulting in a zig-zag that hopefully is as close as possible to the true slope.

That's fine as long as the line has a slope greater or equal to one with its origin at 0,0. Most lines will not, of course, so what we do is to calculate the direction of the true line while normalizing the line to origin 0,0. As in CHIP-8 when you want to subtract 1 from a coordinate, we add $FF to move in a "backward" direction. In binary, we can choose to let $FF = -1 and by adding that value and ignoring the overflow, the coordinate is reduced by one. You may have used the same theory to move a tank in a video game where each of eight hex pad keys chooses a direction for the tank to move.

These directions are kept in MOVTAB (for MOVe TABle) at $01C6.

RF.0 is used as an index into this table to choose the proper direction adders while the line is being normalized into the end point XN,YN. This address is stored on the stack for later as RF is needed for other things and will be changed by POINT which does the job of actually displaying each dot on the line.

Two other quantities are kept. These are A and B and are used to decide whether or not it is time to increase Y_0 for the new X_0 position. (Remember the program always draws a normalized line, but by using appropriate values from MOVTAB, the line actually goes in the intended direction.)

Once it is decided in which direction to move, the appropriate values from MOVTAB are added to the original X_0,Y_0 in RC and also placed in RE to prepare for the call to POINT. Note that there is some duplication in setting RE both initially at $0186 and at $01BB. This simply keeps the loop shorter by two instructions though it does increase the size of the subroutine by two bytes. Also note that even though X is set to F at $01BA, it is reset to 2 by CALL POINT. Not knowing this, a walk-through of the subroutine would appear faulty.

USE AND CARE OF LINE

To connect any two X,Y coordinate pairs, the following steps must be programmed.

1) Set RC.0 = X_0, RC.1 = Y_0 (from points)

2) Set RD.0 = X_1, RD.0 = Y_1 (to points)

3) CALL LINE

A line will be drawn <u>from</u> X_0,Y_0 <u>to</u> X_1,Y_1. It is important to realize that this line may not be exactly equivalent to the line X_1,Y_1 to X_0,Y_0. In other words, if you try to erase a line in the opposite direction it was drawn, you may only partially erase the line. You can't go up the down stairs here. ($012D must be set to $F3 XOR to selectively erase lines without clearing the entire display.)

The subroutine uses a lot of registers, changing R9 through RF except for RB.1 Both RC and RD will also change, so if you will need to use these endpoints later, you must keep them somewhere in memory. The stack is the usual place.

On return from LINE, RC will equal the previous value of RD. That is, RC is set to the other end of the line. To draw a shape of connected lines, then, RC needs to be set only one time with subsequent points set into RD. (See SHAPE sub for more details.)

Please notice that LINE calls POINT and if you are relocating the POINT sub, the CALL at $018A needs to be changed. Other than that, LINE is page relocatable and may be burned into a ROM.

Good luck with your programming. But then... you've heard <u>that</u> LINE before.

Two Color Display Board

by Bob Casey

Those of you who have a color TV but no color board have probably been less than thrilled with a plain black and white image the VIP generates. A simple add-on board can be built to create a colored image - greenish yellow for white, and purplish-blue background for black background. This is not as versatile as the VIP color board, but will generate a more pleasing display than black and white on a color TV. You'll also gain a little experience on color encoding for the American (NTSC) color system.

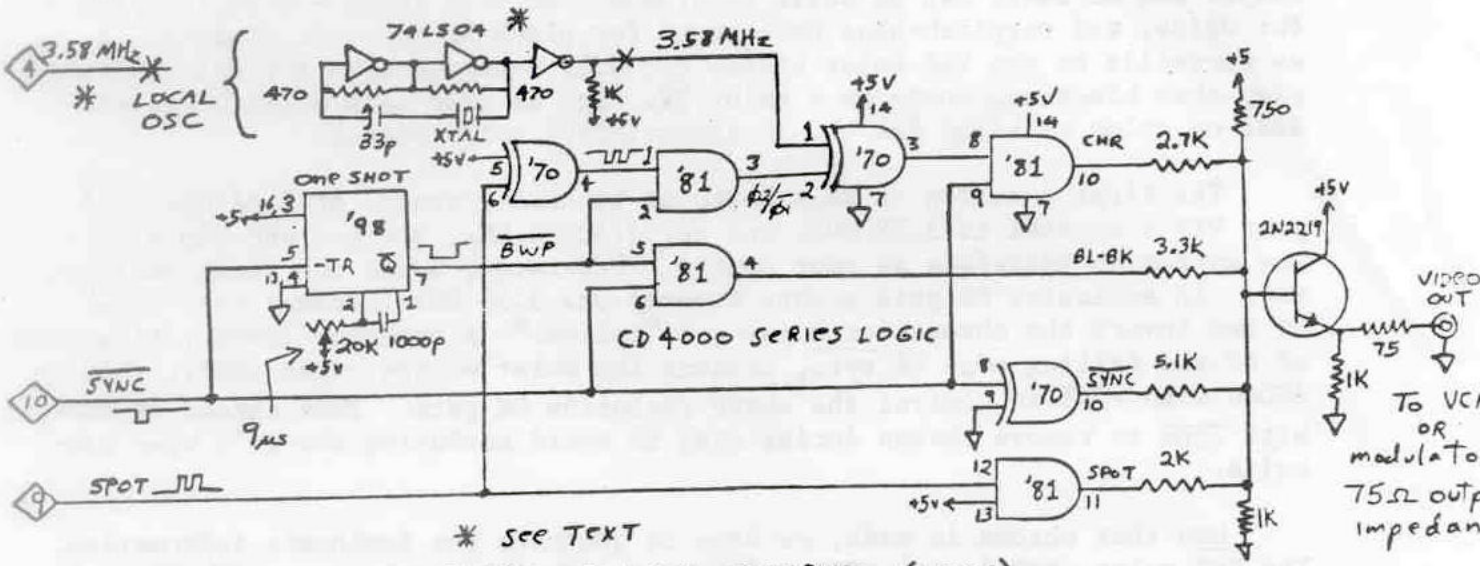
The first function on this board is to have a source of 3.58 MHz. If your VIP's crystal is 3.58 MHz, and not 3.52128 MHz, you can use pin 4 of the expansion interface as your source. Otherwise, build the local oscillator. An exclusive OR gate either inverts the 3.58 MHz (chroma) for "blue" or not invert the chroma for burst and "yellow." A one-shot (4098), triggered of of the falling edge of $\overline{\text{sync}}$, creates the burst window pulse (BWP), then is ANDed with $\overline{\text{spot}}$ to control the above exclusive OR gate. This chroma is ANDed with $\overline{\text{sync}}$ to remove chroma during sync to avoid confusing the TV's sync circuits.

Now that chroma is made, we have to generate the luminance information. The BWP pulse, ANDed with $\overline{\text{sync}}$, will become the blue brightness (BL-BK). It was ANDed to keep it out of the vertical sync time. The spot signal adds brightness for yellow. Left over gates form buffers and inverters.

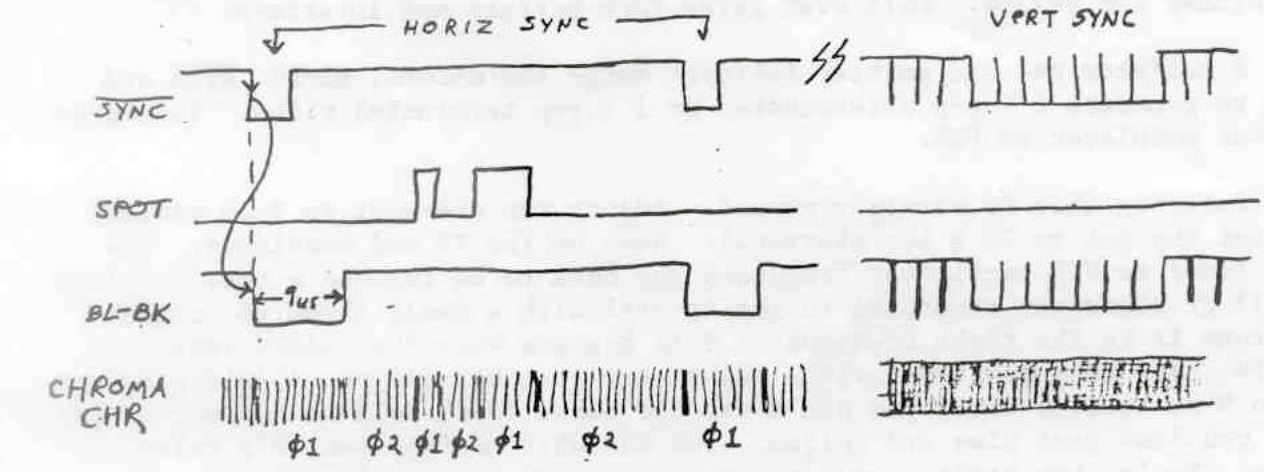
A resistor net and emitter follower merge the chroma, BL-BK, $\overline{\text{sync}}$ and spot to generate 2 V p-p unterminated or 1 V p-p terminated video. This goes to your modulator or VCR.

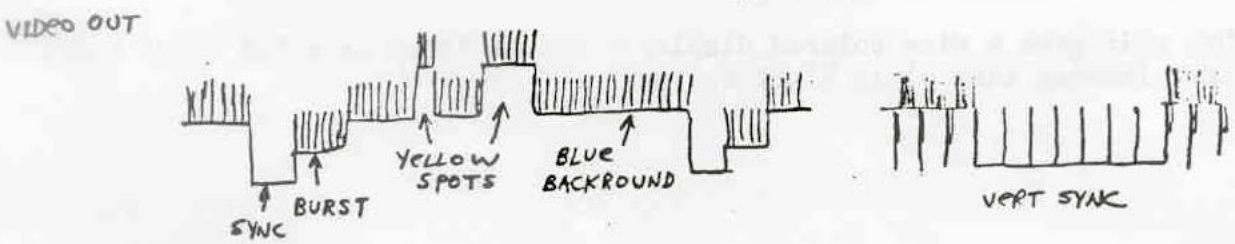
Building this is straightforward. Adjust the one-shot to 9 µs timing (or set the pot to 20 K for starters). Hook up the TV and modulator. The 3.58 local or VIP oscillator frequency may have to be tweaked a bit. Replace the 33 pf capacitor connected to the cyrstal with a small 15-60 pf trimmer and tune it to the right frequency. This happens when the colors appear on the TV. Now that you have color, adjust the one-shot (if you didn't preset it to 9 µs before) until you don't see the burst (dark yellow) but not so far that you lose your blue and yellow. Too far will confuse the TV's color circuits. You're now ready to run programs (no mods required).

This will give a nice colored display - not as fancy as a VIP color board but better looking than plain black and white.



DUAL COLOR OUTPUT (NTSC)





SHAPE OF THINGS TO COME

Though you may use the LINE sub to construct various patterns, you will find SHAPE handy for manipulating objects graphically. Instead of a lengthy machine language routine to draw all the lines of your shape, you simply keep all the endpoints of all the lines in a table. SHAPE will connect every point in the table by drawing a LINE between each one. It's sort of a "connect the dots" subroutine.

The SHAPE TABLE is typically referred to as a matrix. Constructing a SHAPE table could not be simpler -- you only need to enter the X,Y coordinates representing the endpoints of lines. Here is the general format.

SHAPE TABLE FORMAT

X_0,Y_0 -From starting point
X_1,Y_1 -To endpoint #1
X_2,Y_2 -To endpoint #2
: :
: :
: :
XN,YN -To endpoint #N
FF FF -Stop byte

The first X,Y pair is the starting X,Y point followed by each successive point to be connected by a continuous line. The last two bytes in the table are set to $FFFF to signal that this

is the end of the shape. Actually only the most significant bit (MSB) of the second byte needs to be set equal to one to mark the end of the shape table. For example, $0080 would also work. It is easier to use $FFFF however and this provides a visual reference for the end of a table. Some programs in this book and some contemplated VIPER contributions, will require an $FFFF stop byte. (If you happen to have a copy of PIPS I, by the way, the CHIP-8 Editor provides a handy way to enter shape tables and move them around in memory.)

USE AND CARE OF SHAPE

The following steps must be programmed to draw a shape on the screen:

1) Enter a shape table in the proper format anywhere in memory
2) Set R7 to the address of the shape table
3) CALL SHAPE

After the shape is drawn, R7 is left pointing to the byte immediately following the $FFFF stop indicator. If shape tables are kept one after the other in memory, repeated calls to SHAPE may be made in order to draw all the shapes without resetting R7 each time.

Well, that's about the SHAPE of it!*

*It FIGURES, doesn't it?

THE PLOT THICKENS

PLOT works identically to SHAPE except that the points in the shape table are simply plotted instead of connected. The same format for a shape table is used (see SHAPE). Also the same ending conditions for R7 exist following a call to PLOT.

They say every good book must have a good PLOT, you know.

WHAT A FLOP

If you have read the chapter on page switching graphics for CHIP-8 programs, you already know how to use the FLOP subroutine. But read on, the technique is a little different in machine language.

The graphics subroutines on the tape and in the listing that follows this chapter must be modified to enable the FLOP sub. You must make these modifications by using the "write mode" of your system monitor before calling FLOP for the first time.

```
0121  FB 04  XRI  $4 -- POINT sub change
0131  FB 07  XRI  $7 -- CLEAR sub change
```

That's all you have to do to enable the Graphics Subs Package #1 for page switching. Now when you call the POINT sub or the CLEAR sub (or LINE or SHAPE or PLOT, too which use POINT) the actual graphics are inserted into the display page not shown on the screen. CALLing FLOP will bring that page into view. The

taped version of Graphics Subs Package #2 contains the above modifications.

USE AND CARE OF FLOP

The procedure for using FLOP is similar to its use in CHIP-8 as previously discussed. Remember to make the necessary modifications to CLEAR and POINT before calling FLOP, however. It's best to keep two versions of the graphics subroutines -- one without FLOP and one with FLOP enabled. Later, the demonstration programs will call for one version or the other.

To use FLOP during a program, first execute the following sequence:

```
1) CALL FLOP ;FLOP to cleared page
2) CALL CLEAR ;CLEAR other page
```

Because the initialization sequence in the graphics package always calls CLEAR one time, the above sequence will result in the entire eight memory page buffer being cleared.

Your graphics programming may use the following set up to animate a display.

```
3) CALL FLOP ;View "static" page
4) CALL CLEAR ;Erase the other page
```

5) Display shapes, ;Your programming
   lines, points ; goes here
   etc.
6) GOTO #3 ;Loop to view new graphics

Numbers 3 and 4 may be combined with 1 and 2 above if convenient to the program.

To selectively animate objects without clearing the whole screen is not quite so simple. Erasing an object requires knowing its previous position, not its current one. To move a ball to the left, for example, you must erase the old ball at (X - 1,Y) then redraw the new ball at (X + 1,Y). For animating complex shapes, the following formula must be used: (Note that $012D must be set to $F3 XOR for this to work. The OR instruction in POINT would prohibit selective erasing of figures on display.)

```
DX = change in X i.e. amount to be added to X
DY =    "   "  Y  "     "    "  "    "  "  Y

ERASE FIGURE AT:
        ((X-DX),(Y-DY))

DRAW FIGURE AT:
   X,Y = ((X+DX),(Y+DY))
```

That, unfortunately, doesn't solve all the problems. Because there are now two display screens, both must be initialized before

trying to erase a figure. You can't erase something that has not yet been drawn. In addition to the above formulas the following will get the display set up properly and move any object. (Assume both display refreshes to be empty.) Remember that the erasing and drawing is going on behind the curtain on the display page you <u>can't</u> see.

```
1) Draw figure at X,Y          ;Initialize one display
2) CALL FLOP                   ;View other page
3) Let X=(X+DX), Y=(Y+DY)      ;Calculate new X,Y
4) Draw figure at X,Y          ;Animate
5) CALL FLOP                   ;View animation
6) Erase figure at             ;Do not change value
     (X-DX), (Y-DY)              of X,Y!
7  GO TO  #3                   ;Loop
```

To follow the logic of this algorithm (meaning method or proposed schematic for a program), it may help for you to work out the values of X and Y for a simple display. Let X = Y = DX = DY = 1 and "walk through" the above steps until you see how the animation procedes. Notice that it is important that X,Y not be changed to erase old figures in #6. Only in #3 are the values of X,Y actually changed.

Games that need to know if one figure has touched another may insert the following into the above scheme.

5.1) If no hit, go to #6

5.2) Erase figure at (X-DX),(Y-DY) ;Same as #6

5.3) CALL FLOP

5.4) Erase figure at X,Y ;Both figures erased

5.5) Do appropriate "HIT" sequence

5.6) Exit or go to #1 for restart

The same algorithm may be used to selectively animate objects in CHIP-8 programs using the page switching techniques discussed earlier. At no time is it important to know which display refresh page is being viewed and which is the "work page." The flopping is completely automatic and transparent to both the program and the viewer.

You may call FLOP without having enabled the sub by modifying POINT and CLEAR as said earlier. The effect of this is to have two display refresh pages, one at $0800 - $0BFF and one at $0C00 - $0FFF, which may be used as regular display areas. You can FLOP back and forth between pages but all erasing and drawing will occur on the page currently being viewed. In fact, if you want to watch the behind screen display action during debugging of a page switching animation, this technique sort of hands you a backstage pass. And by rapidly flopping between the two pages, two displays may be multiplexed -- that is "combined" -- into one apparent display.

Page switching is an advanced and highly sophisticated

graphics technique used by some of the fanciest computer programs on the market. An understanding of the logic behind the method will help you to use it effectively and it is strongly suggested you "think it through" several times. You'll save yourself some debugging time trying to figure which came first, the FLIP or the FLOP.

FLOP to it!

"CH-CH-CH-CH-CHANGES"

CALL CHANGE to flip to different display resolutions from within your program. The normal method of first turning off the display (or disabling interrupt acknowledgements), resetting R1 to the new interrupt routine address, then turning the display back on is not necessary. Also, the display will not flicker when CHANGing as it would with the other method.

CHANGE works by taking advantage of the interrupt timing. Just after an interrupt occurs, we can be sure that a length of time will pass before the next one. A simple test routine that incremented a counter between two interrupts showed 262 instructions being executed from the end of one interrupt routine to the beginning of the next interrupt cycle. The instructions in the interrupt routine were not counted.

This indicates that there is plenty of time to change register R1 to a new interrupt routine address without the worry that an interrupt will occur halfway during the process of setting R1 to

a new value. The IDL instruction at $00C5 causes the 1802 processor to wait at that address until an interrupt happens. Therefore, the instruction at $00C6 will be the first instruction to be performed following the interrupt and the section at $00C6 to $00CA will fall into the gap between interrupts allowing R1 to be set to a new address.

The six bytes at $00CB - $00D0 may hold the addresses of up to three different interrupt routines. In the listing here, the two page and four page routines have been entered along with the VIP's ROM one page interrupt routine at $8146.

To use CHANGE, RE.0 is first set to a value of 0,1 or 2 and the subroutine is called. If RE.0 = 0, then the first interrupt routine in RESTAB (for interrupt RESolution TABle) will be enabled, etc. Because a one page display is of limited use with these graphics subs, it was included more "because it was there" than for any other reason. The VIP's ROM routine changes both R8 and R9 as noted in the VIP manual and may conflict with the graphics subs register use and with some of the programs later in this book. If you want a permanent one page display, it is suggested that you write a new interrupt routine for this purpose and place its address in RESTAB to avoid a register conflict.

Be sure not to set RE.0 to a value greater than two or CHANGE will look beyond the end of RESTAB for a new interrupt routine address and I don't have to tell you what that could mean. It would probably be a CHANGE for the worse.

GRAPHICS SUBS REFERENCE

Here is a handy, quick reference to each of the machine language graphics subs. All input and output conditions are specified along with which registers will be changed during the action of a subroutine. The CALL address is used in your program by executing a SEP R4 ($D4) instruction followed by the routine's address. For example, the three bytes D4 01 47 would call the LINE sub at $0147. Before doing this, the endpoints of the line would have to be stored in RC and RD as noted under the INPUT conditions to LINE.

Notice that some subroutines call others in the package. You must be careful to consider all the possible register changes for those routines. SHAPE, for instance, only lists R7, RC and RD as the changing registers. But SHAPE also calls LINE (which also calls POINT) and you must be aware of register uses in those subs as well.

```
CHANGE:   CALL      $00BD
          INPUT:    RE.0 = 0,1,2 for resolution #0, 1 or 2
          OUTPUT:   R1 set to new interrupt routine address
          CHANGES:  R1, RF

CLEAR:    CALL      $0130
          INPUT:    RB.1 = display page
          OUTPUT:   Four pages erased
          CHANGES:  RE.0, RF
```

```
FLOP:     CALL        $007F
          INPUT:      RB.1=one display page pair in 2K boundaries
          OUTPUT:     RB.1 = other display page
          CHANGES:    RB.1

          (Also change: 1) 0121  FB 04  XRI  $4
                           0131  FB 07  XRI  $7
           before calling FLOP first time!)

LINE:     CALL        $0147
          INPUT:      RC.0 = X0, RC.1 = Y0 (from points)
                      RD.0 = X1, RD.1 = Y1 (to points)
          OUTPUT:     Line drawn from X0,Y0 to X1,Y1
                      RC.0 = X1, RC.1 = Y1 on return
          CALLS:      POINT
          CALLED BY:  SHAPE
          CHANGES:    R9, RA, RB.0, RC, RD, RE, RF

MERGE:    CALL        $0400
          INPUT:      R7 addresses matrix #1
                      R8 addresses matrix #2
          OUTPUT:     Matrix #1 merges into matrix #2
                      R7 undefined
                      R8 not changed
          CALLS:      CLEAR   SHAPE   FLOP
          CHANGES:    R7, RE, RF
 (NOTE: MERGE included here for your convenience.  This routine
  is discussed later.)

PLOT:     CALL        $0098
          INPUT:      Same as SHAPE
          OUTPUT:     Points plotted from X,Y matrix
                      R7 ends as in SHAPE
          CALLS:      POINT
          CHANGES:    R7, R8

POINT:    CALL        $0108
          INPUT:      RE.0 = X, RE.1 = Y, RB.1 = display page
          OUTPUT:     Point plotted at X,Y
          CALLED BY:  LINE   PLOT
          CHANGES:    RE, RF

SHAPE:    CALL        $0084
          INPUT:      R7 addresses matrix of X,Y coordinates
          OUTPUT:     Coordinates connected by lines
                      R7 addresses byte after table (ending FFFF)
          CALLS:      LINE
          CHANGES:    R7, RC, RD
```

LISTING

REGISTER ASSIGNMENT

R0 - DMA pointer set and changed by interrupt action
R1 - Interrupt routine program counter
R2 - Stack pointer. Page 2 ($0200-$02FF) reserved for stack
R3 - Normal program counter at all times
R4 - CALL routine program counter
R5 - RETN routine program counter
R6 - Pointer to arguments. Holds return addresses
R7 - Available (But needed as a pointer in SHAPE and PLOT)
R8 - Available
R9 - Work registers as noted for each graphics sub
RA - " " "
RB - " " "
RC - " " "
RD - " " "
RE - Usually available for immediate uses
RF - " " "

MEMORY MAP (4K SYSTEMS) NORMAL

0000 - 0024 - Initialization
0025 - 007E - Available for short programs (see notes following)
007F - 01E5 - Graphics Subroutines
01E6 - 01FF - Available
0200 - 02FF - Stack (use upper regions of page with care)
0300 - 0BFF - 2304 bytes available for programming
0C00 - 0FFF - Display refresh

MEMORY MAP (4K SYSTEMS) FLOP ENABLED

0000 - 02FF - Same as above
0300 - 07FF - 1280 bytes available for programming
0800 - 0BFF - Display refresh #1
0C00 - 0FFF - Display refresh #2

SUBROUTINE ADDRESSES

007F - 0083 - FLOP (CALL $007F)
0084 - 0094 - SHAPE (CALL $0084)
0095 - 009F - PLOT (CALL $0098)
00A0 - 00AF - CALL
00B0 - 00BC - RETN
00BD - 00D0 - CHANGE(CALL $00BD)
00D1 - 00EF - INT2
00F0 - 00FF - INT4
0100 - 012F - POINT (CALL $0108)
0130 - 0145 - CLEAR (CALL $0130)
0147 - 01E5 - LINE (CALL $0143)

```
                    ;Initialize Computer

0000  90          BEGIN: GHI  R0      ;Get ØØ byte from R0.1
  01  B3                 PHI  R3      ;R3.1=00 (Program counter)
  02  B4                 PHI  R4      ;R4.1=00 (CALL PC)
  03  B5                 PHI  R5      ;R5.1=00 (RETN PC)
  04  F8 02              LDI  STACK.1;Page address for stack (02 normally)
  06  B2                 PHI  R2      ;R2.1=02
  07  F8 FF              LDI  $FF
  09  A2                 PLO  R2      ;R2=02FF=stack base address
  0A  F8 A1              LDI  CALL.0
  0C  A4                 PLO  R4      ;R4=00A1=CALL sub entry point
  0D  F8 B1              LDI  RETN.0
  0F  A5                 PLO  R5      ;R5=00B1=RETN sub entry point
0010  F8 0C              LDI  DISP.1  ;(2K=$04/3K=$08/4K=$0C)
  12  BB                 PHI  RB      ;RB.1=start of display page (0C normal
  13  F8 17              LDI  START.0;
  15  A3                 PLO  R3      ;Set up R3 to become PC
  16  D3                 SEP  R3      ;PC=R3 freeing RØ for DMA
  17  D4 01 30  START:   CALL CLEAR   ;Clear display page before video on
  1A  F8 00              LDI  INT4.1  ;Set R1 to address of 4-page
  1C  B1                 PHI  R1      ;  (or other) interrupt routine
  1D  F8 F2              LDI  INT4.0  ;    "          "          "
  1F  A1                 PLO  R1      ;    "          "          "
0020  69                 INP  DEV-9   ;Turn on video (X is 2 from CALL)
  21  D4 03 00           CALL PROG    ;User program is a subroutine
  24  23                 DEC  R3      ;Halt.  In case too many
                                      ;  RETN's executed, program
                                      ;  will halt here preventing a
                                      ;  common cause of crashes.

  25  00           (PROG:)            ;Usually located at $0300,
                                      ;  short programs may start here
                                      ;  by changing the address of
                                      ;  the CALL in $0021.


                    ;FLOP Sub

  7F  9B           FLOP: GHI  RB      ;Get current display page
0080  FB 04             XRI  $4      ;Exclusive OR with $4 to flop (08/0C)
  82  BB                PHI  RB      ;Put new value back in RB.1
  83  D5                RETN         ;Return

                    ;NOTE--Must also change:
                    ;  1) 0121  FB 04  XRI $4
                    ;  2) 0131  FB 07  XRI $7
                    ;before calling FLOP!
```

```
                    ;SHAPE Sub

0084  47          SHAPE: LDA  R7
  85  AC                 PLO  RC      ;RC.0←X (start)
  86  47                 LDA  R7
  87  BC                 PHI  RC      ;RC.1←Y (start)
  88  30 8D              BR   2F      ;Branch to begin loop
  8A  D4 01 47     1H:   CALL LINE    ;Plot line from RC to RD
  8D  47           2H:   LDA  R7
  8E  AD                 PLO  RD      ;RD.0←X (to)
  8F  47                 LDA  R7
0090  BD                 PHI  RD      ;RD.1←Y (to)
  91  FE                 SHL          ;Test for bit set (i.e.>7F)
  92  3B 8A              BNF  1B      ;If not, branch to loop
  94  D5                 RETN         ;Exit on stop byte>7F

                    ;PLOT Sub

0095  D4 01 08     1H:   CALL POINT
  98  47           PLOT: LDA  R7
  99  AE                 PLO  RE      ;RE.0←X
  9A  47                 LDA  R7
  9B  BE                 PHI  RE      ;RE.1←Y
  9C  FE                 SHL          ;Test for bit set (i.e.>7F)
  9D  3B 95              BNF  1B      ;Branch to loop
  9F  D5                 RETN         ;Exit on stop byte>7F

                    ;CALL Sub -- Runs in R4

00A0  D3           1H:   SEP  R3      ;Exit - call sub with R3=PC
  A1  E2           CALL: SEX  R2      ;X=2
  A2  96                 GHI  R6      ;Push return address of last
  A3  73                 STXD        ;  sub call in R6 onto stack
  A4  86                 GLO  R6      ;   "          "          "
  A5  73                 STXD        ;   "          "          "
  A6  93                 GHI  R3      ;Get return address from former
  A7  B6                 PHI  R6      ;  PC R3 and place in R6
  A8  83                 GLO  R3      ;   "          "          "
  A9  A6                 PLO  R6      ;   "          "          "
  AA  46                 LDA  R6      ;Using R6 as a pointer, pick up
  AB  B3                 PHI  R3      ;  subroutine address from where
  AC  46                 LDA  R6      ;  CALL was issued and place
  AD  A3                 PLO  R3      ;  in R3
  AE  30 A0              BR   1B      ;Branch to exit, resetting R4
                                      ;  for next CALL
```

```
                        ;RETURN Sub -- Runs in R5

00B0  D3          1H:   SEP  R3      ;Exit - Return via R3=PC
 B1   96         RETN:  GHI  R6      ;Get return address from R6
 B2   B3                PHI  R3      ;  and place in R3
 B3   86                GLO  R6      ;   "    "     "
 B4   A3                PLO  R3      ;   "    "     "
 B5   E2                SEX  R2      ;X=2
 B6   60                IRX          ;Point to return address on stack
 B7   72                LDXA         ;Pop address from stack and
 B8   A6                PLO  R6      ;  place in R6 for next RETN
 B9   F0                LDX          ;   "     "     "    "
 BA   B6                PHI  R6      ;   "     "     "    "
00BB  30 B0             BR   1B      ;Branch to exit, resetting R5


                        RESOLUTION CHANGE

00BD  F8 00     CHANGE: LDI RESTAB.1;Load high address of interrupt
 BF   BF                PHI  RF      ;  addresses table into RF.1
00C0  8E                GLO  RE      ;Get passed index number
 C1   FE                SHL          ;Multiply X 2 and add low
 C2   7C CB             ADCI RESTAB.0; address of table to form address
 C4   AF                PLO  RF      ;RF now addresses RESTAB
 C5   00                IDL          ;Wait for interrupt to occur
 C6   4F                LDA  RF      ;Now quickly transfer
 C7   B1                PHI  R1      ;  address in table via RF
 C8   0F                LDN  RF      ;  to R1 so that the new
 C9   A1                PLO  R1      ;  resolution will run on
 CA   D5                SEP  R5      ;  the next interrupt.


00CB  81 46     RESTAB: .WORD $81,$46;Address 64 x 32 resolution
 CD   00 D3             .WORD INT2   ;Address 64 x 64 resolution
 CF   00 F2             .WORD INT4   ;Address 64 x 128 resolution

                        ;NOTE--Insert addresses of interrupt
                        ;  routines where ".WORD" is.  First
                        ;  address is resolution #0, second
                        ;  is resolution #1, etc.  Do not
                        ;  set RE.0 greater than the number
                        ;  of addresses in RESTAB!
```

TWO PAGE RESOLUTION INTERRUPT

```
00D1  72        INTRET: LDXA          ;Pop saved "D" value into D
  D2  70                RET           ;Return from interrupt.  Set IE=1
  D3  C4          INT2: NOP           ;3 cycle no operation for sync
  D4  22                DEC  R2       ;Point to free stack location
  D5  78                SAV           ;Push the X & P registers via T
                                      ;  register
  D6  22                DEC  R2       ;Point to free stack location
  D7  52                STR  R2       ;Push value of "D" register
  D8  9B                GHI  RB       ;Get start page address in RB.1
  D9  B0                PHI  R0       ;Put in R0.1 to prepare for DMA
  DA  F8 00             LDI  #00      ;Also set R0.0 to 00 but may
  DC  A0                PLO  R0       ;  be any value for midpage start
  DD  C4               *NOP           ;Eight cycles for sync.  May be
  DE  C4               *NOP           ;  any eight cycle instruction
  DF  E2               *SEX  R2       ;  combination
00E0  80          DISP: GLO  R0       ;Get value DMA pointer to save
  E1  E2               *SEX  R2       ;NOP- 8 DMA bursts occur just before
                                      ;  here
  E2  20                DEC  R0       ;Reset DMA pointer in case past page
  E3  A0                PLO  R0       ;Reset pointer to same byte (repeat
                                      ;  2 times)
  E4  E2               *SEX  R2       ;NOP- 8 DMA bursts occur just before
                                      ;  here
  E5  3C E0             BN1  DISP     ;If not EF1, branch.  Else exit loc
  E7  80        DISEF:  GLO  R0       ;Repeat above loop to display
  E8  E2               *SEX  R2       ;  last four lines of display
  E9  20                DEC  R0       ;The "SEX R2's" are NOPs
  EA  A0                PLO  R0       ;
  EB  E2               *SEX  R2       ;
  EC  34 E7             B1   DISEF    ;If still EF1, branch to repeat
  EE  30 D1             B2   INTRET   ;Else branch to exit
```

FOUR PAGE RESOLUTION INTERRUPT

```
00F0  72        INTRET: LDXA          ;Pop saved D value into D
  F1  70                RET           ;Return from interrupt.  Set IE=1
  F2  C4          INT4: NOP           ;3 cycle no operation for sync
  F3  22                DEC  R2       ;Point to free stack location
  F4  78                SAV           ;Push X & P registers via T register
  F5  22                DEC  R2       ;Point to free stack location
  F6  52                STR  R2       ;Push value of D register
  F7  E2               *SEX  R2       ;No operation
  F8  E2               *SEX  R2       ;No operation
  F9  9B                GHI  RB       ;Get display page start from RB.1
  FA  B0                PHI  R0       ;Put in R0.1 to set for DMA
  FB  F8 00             LDI  #00      ;Also set R0.0 to 00 but may be
  FD  A0                PLO  R0       ;  changed for midpage starts
  FE  30 F0             Br   INTRET   ;Branch to exit
```

```
                ;New Year's Resolution

NYRRES: LDN   R0      ;Load user's statement
        STR   CLOCK   ;Store behind the clock on mantle
  1H:   INC   TIME
        INC   TIME    ;Give user a chance to
        INC   TIME    ;  comply
        XRI   REALITY;Compare with user's actions
        BZ    1B      ;If equal, loop
        INC   BLKMRK  ;Else give user a black mark
        GLO   BLKMRK  ;
        XRI   LIMIT   ;Test for failure
        BNZ   1B      ;Branch on< failure
        JMP   GUILTY  ;Jump on failure --
                      ;  pay the consequences
```

*NOTE: Asterisk marked instructions may be replaced by others to take advantage of the interrupt timing. A very simple "random number" generator may be had by incrementing a register at one of the asterisk locations. When the number will be requested by human input, the returned value should be sufficiently random for games though for scientific applications, such a procedure would be grossly inadequate. This will work best at locations 0N10 and 0N13 taking the low eight bits of the register for the pseudo random number. At these points the register will be incremented 60 times <u>every</u> 1/60 seconds or 3600 times a second.

```
                ;POINT Sub

0100            BITTAB: .BYT $80,$40,$20,$10
  04                    .BYT $08,$04,$02,$01
  08  8E         POINT: GLO  RE     ;Get X coordinate from RE.0
  09  F6                SHR        ;Shift right three times to divide
  0A  F6                SHR        ;  by 8 forming relative
  0B  F6                SHR        ;  horizontal display address
  0C  52                STR  R2    ;Push onto stack for later
  0D  8E                GLO  RE    ;Get X coordinate from RE.0
  0E  FA 07             ANI  $7    ;Save 3 LSB's for bit in byte
                                   ;  position
0110  AE                PLO  RE    ;Put in RE.0 for addressing
                                   ;  BITTAB later
  11  9E                GHI  RE    ;Get Y coordinate from RE.1
  12  FE                SHL        ;Multiply x 2 (overflow not possible)
  13  FE                SHL        ;Now times 4 and overflow is possible
  14  AF                PLO  RF    ;Store in RF.0 temporarily
  15  F8 00             LDI  $0
  17  7E                SHLC       ;Add possible carry to 00 byte
```

```
0118  BF              PHI  RF     ;Put in RF.1 temporarily
 19   8F              GLO  RF     ;Finish multiply x 8 by shifting
 1A   FE              SHL         ;  RF.0 once more to the left
 1B   F1              OR          ;OR in horizontal address on stack
 1C   AF              PLO  RF     ;Return finished low address to
                                  ;  RF.0
 1D   9F              GHI  RF     ;Finish multiply x 8 by shifting
 1E   7E              SHLC        ;  RF.1 once more left with
                                  ;  possible carry
 1F   52              STR  R2     ;Push this value onto the stack

0120  9B              GHI  RB     ;Get current display page address
 21   E2              SEX  R2     ;(NOP) Save 2 bytes for page
 22   E2              SEX  R2     ;(NOP) switching techniques
 23   74              ADC         ;Add base page to offset
 24   BF              PHI  RF     ;RF addresses display byte
 25   93              GHI  R3     ;Get BITTAB.1 from program counter
 26   BE              PHI  RE     ;RE addresses proper display bit
                                  ;  pattern
 27   0E              LDN  RE     ;Load byte with single bit set
 28   52              STR  R2     ;Push for inserting into display
 29   0F              LDN  RF     ;Load byte in display refresh
 2A   F2              AND         ;Logical AND to test if bit set
 2B   AE              PLO  RE     ;Flag possible "Hit" (00=no hit/
                                  ;  ≠00=hit)
 2C   0F              LDN  RF     ;Reload display byte
 2D   F1              OR          ;Combine with bit on stack
 2E   5F              STR  RF     ;Return new display byte to display
 2F   D5              SEP  R5     ;Return from subroutine


                ;CLEAR PAGE Sub

0130  9B       CLEAR: GHI  RB     ;Get display page address from RB.1
 31   FC 03           ADI  $3     ;Add 3 for last page
 33   BF              PHI  RF     ;Put in RF.1 as a work pointer
 34   F8 FF           LDI  $FF    ;Set RF.0=$FF so RF addresses
 36   AF              PLO  RF     ;  last byte of last page
 37   F8 04           LDI  $4     ;Set RE.0=4 as a loop count
 39   AE              PLO  RE     ;  to erase 4 memory pages
 3A   EF              SEX  RF     ;X=F to facilitate erasing
 3B   F8 00      1H:  LDI  $0     ;Load 00 byte for erasing
 3D   73              STXD        ;Store via RF (X=F) to erase a byte
 3E   8F              GLO  RF     ;Test if RF.0 is zero yet
 3F   3A 3B           BNZ  1B     ;If not, loop to continue erasing
0141  2E              DEC  RE     ;Count # times RF crosses a page
 42   8E              GLO  RE     ;Test if RE.0 is zero yet
 43   3A 3B           BNZ  1B     ;If not, loop to erase next page
 45   5F              STR  RF     ;Erase last byte (Keeps loop fast)
 46   D5              RETN        ;Return from subroutine
```